Automating Component Test Insertion into DØRunII CTBUILD Software Packages

Mariano A. Zimmler

New York University
New York, NY 10012

Supervisor
Dr. David J. Ritchie
Computing Division

Fermi National Accelerator Laboratory
Summer Internships in Science and Technology
Batavia, IL 60510

Summer 2002

1 Abstract

This paper describes two projects related to the DØ Experiment code releases. The first and major project involved writing a meaningful test for every component in each DØRunII CTBUILD software package. The second project consisted in writing a script that would generate one or more web-pages displaying the status of the DØ code releases.

2 Introduction

The DØ experiment was proposed for the Fermilab antiproton-proton Tevatron Collider in 1983 and approved in 1984. After 8 years of design, testing, and construction of its hardware and software components, the experiment recorded its first antiproton-proton interaction on May 12, 1992, starting the data-taking period referred to as "Run 1" which would last until the beginning of 1996. During this period, antiproton-proton collisions were studied mainly at an energy of 1800 GeV in the center of mass.

For many years, our understanding of nature has revolved around four separate and seemingly unrelated forces: gravity, the electromagnetic force, the weak force, and the strong force. Over the last three decades, the development of many experimental and theoretical advances has led to a coherent and predictive picture of nature that could successfully account for all of these forces with the sole exception of gravity—the Standard Model (SM).

During the 1960s and 70s, it was recognized that the electromagnetic and weak forces could be unified under a single description, and the theory of electroweak interactions was born. The strong and electroweak interactions, however, still remained apart. There are now compelling reasons to believe that the Standard Model, though remarkably accurate in its predictions, is still only an approximate description of nature. Many other theories have been postulated that provide unification of the forces but they have still not been tested.

According to the Standard Model, the particles created at the Tevatron fall into two broad classes: leptons (electrons, muons, taus, and a different flavor of neutrino associated with each of them) and hadrons (protons, pions, kaons, etc.). These are simply combinations of the six quarks. Each quark and lepton is also mirrored by its corresponding antiparticle. The gauge bosons transmit the

fundamental forces; these include the photon (electromagnetic force), the gluons (strong force), and the W and Z bosons (weak force).

The Fermilab accelerator complex provides extremely high intensity proton and antiproton beams at the world's highest energy (900 billion electron volts) that can probe these fundamental forces. These two beams collide at several locations around the Tevatron ring. At two of these locations, experiments are performed by the CDF and DØ collaborations.

The DØ experiment contains many sophisticated components, besides just the particle detectors. These include the electronics needed to select and digitize events and the software necessary to monitor the experiment and reconstruct events written to magnetic tape.

The DØ detector is composed of three major subsystems: a collection of tracking detectors extending from the beam axis to a radius of 30 inches, energy-measuring calorimeters surrounding the tracking region and an encompassing muon detector that deflects muons using solid iron magnets. The entire detector is about 65 feet long, 40 feet wide and high, and weighs approximately 5500 tons.

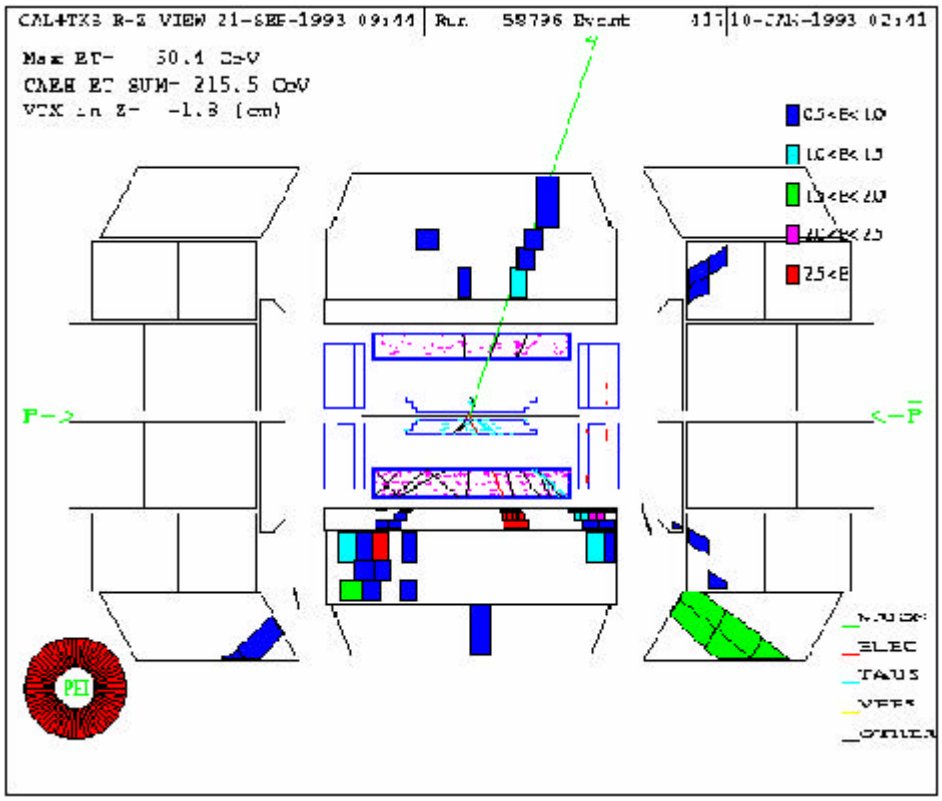

Fig. 1: A side view of a reconstructed event. The muon track is shown as a green line, the electron track is shown as a short red line, and the two main jet energy depositions in the calorimeters are shown in different colors that represent the energies in the contributing cells.

The detector has over 120,000 channels of individual electronic signals. These are used to analyze the properties of an event and to decide whether it is a candidate for further study. This "triggering" occurs in three stages: the first stage is completed within 4 microseconds, before the next accelerator beams bunch arrives at the detector. A second stage follows the digitization of all the information in

a farm of dedicated microprocessors. Events that are interesting enough to survive this process are eventually written to tape and reconstructed for subsequent analysis.

The figure above shows a characteristic event observed in the detector. The directions of all charged particles are measured in tracking chambers surrounding the collision point. These detectors rely upon the ionization of a gas caused by the passage of charged particles; the produced ionization is focused electrically onto sensors that record the amount of charge and its time of arrival, and thus permit the reconstruction of the particle trajectory.

The energy of all particles except muons and neutrinos is measured in the three calorimeters surrounding the tracking volume. Each is composed of a stack of heavy metal plates (uranium, steel or copper) interspersed between gaps containing liquid argon. Particles hitting on the calorimeters interact with it, yielding secondary particles, which also interact, leading to a shower of particles that ultimately ends when all the secondary particles lose their energy and stop. The passage of the full set of showering particles through the argon gaps produces ionization electrons that are collected on localized electrodes; the observed signal is then proportional to the incoming particle energy. The pattern of energy deposition along the shower is used to distinguish electrons or photons from hadrons. Clusters of deposited energies are used to reconstruct the jets associated with quarks and gluons.

Muons, however, can penetrate the calorimeters without much change in their energy or direction. To detect them, gas-filled tracking chambers positioned before and after magnetized blocks of iron are placed in the outer region of the detector. These chambers provide the muon trajectories before and after the bend in the magnet, and thus yield the momentum or energy of the muons.

Neutrinos do not interact with matter enough to be detected directly. Although their presence can be inferred from inverse beta decay processes, their energy and momentum is typically derived from the apparent imbalance in the conservation of those quantities in a collision.

The computer software for DØ is almost completely custom-written. It is required for monitoring and control of the experiment, for the microprocessors in the trigger system, for controlling the data flow to the ultimate logging to tape, for the reconstruction of particles from the signals measured in the detector, and for managing the large data samples (70 million events, 3 Terabytes of data) acquired over the Run I. Special attention has been paid to graphical displays of events and detector performance. Many millions of simulated events have been created for the study of detector performance and specific physics processes through "Monte Carlo" programs that mimic the response of the detector. The software that processes both the real and simulated data samples is the one this project is concerned with.

3 Background: The physical structure of DØRunII software

DØ has previously adopted BaBar's SoftRelTools (SRT) as its standard code management environment. However, SRT lacked any support for testing, other than linking test executables. DØ adopted Dave Adam's CTEST build interface specification, implemented by CTBUILD.

The standard code management environment simply defined a convention for the physical organization of both the code and the instructions for building C++ software.

The next few sections provide a detailed overview of relevant characteristics of the CTEST specification necessary to the understanding of the results of the project.

3.1 Code Organization

The fundamental unit of the software organization in CTEST is the *component*, which consists of a *header* file, an *implementation* file and a *test* file. The implementation file contains the source, which is compiled into a bare object or a library. It includes (through the preprocessor directive `#include`) the header. *The test file is a main program which tests the header interface and implementation source and returns zero if the test is succesful.* Typically a component specifies one class and associated free functions.

The DØRunII Software Release consists of over three thousand components. Components are grouped into packages (~500) and packages are grouped into subsystems (~25). The code and instruction files for each package are contained in a separate directory. Each package contributes only to its own library. A typical package might have five to ten components although there are no limitations to the number of components a package can have. CTEST is also defined only at the package level. The instructions for building are specific to each package.

In addition to components, a package may include stand-alone headers, additional tests and sources to be compiled into binaries and scripts. The package directory may be divided into subdirectories for the purpose of organization or to apply different instructions to different pieces of code. However, all the library components must reside in the same subdirectory. Everything in the package directory including subdirectories is considered to be a part of the package and all files which are part of the package are contained in the directory.

3.2 Dependencies

CTEST does not specify the order in which packages are built but it does specify the order of components within a package.

Packages may depend on one another through included header files or through libraries. Headers in other packages are incorporated using an include mechanism that is slightly different from the C++ standard—namely, `#include "package/file"`, where the first directory in the include file name is the package name.

Action	Relevant instruction file
Header installation	HEADER_DIR*, HXXTYPE*, INCLUDE_TYPES*, INCLUDE_FILES*, INCLUDES
Library dependencies	LIBDEPS*
Library build	COMPONENTS, CXXTYPE*
Component testing	COMPONENTS, TXXTYPE*, CTEST_DIR*
Object build	OBJECT_COMPONENTS, CXXTYPE*
Integrated testing	ITEST, OBJECTS, LIBRARIES, CXXTYPE*
Binary build	BINARIES, OBJECTS, LIBRARIES, CXXTYPE*
Script installation	SCRIPTS

Table 1: CTEST actions and associated instruction files. Those marked with asterisks only appear in the top-level package directory.

3.3 Build instructions

The instructions for building a package are taken from instruction files within the package. Table 1 lists the actions which are carried out and the instruction files on which each action depends. Those

instructions that apply package-wide and appear in the top-level package directory are marked with asterisks. The others apply only to the subdirectory in which they reside.

3.3.1 File extensions

There are many conventions for C++ file name extensions and CTEST does not impose one. Instead, the three files HXXTYPE, CXXTYPE and TXXTYPE specify the extensions for header, source implementation and test files, respectively. These files, however, do not appear in any of the DØRunII packages. The respective default .hpp, .cpp or _t.cpp is used. Although the default extension for the header is .hpp, other extensions are also supported such as .h, .hh and .H.

3.3.2 Processing subdirectories

For all actions except header installation, instructions may reside in the top-level directory or any subdirectory. Each of the subdirectories listed in the instruction file SUBDIRS is processed after carrying out the action in the current directory.

3.3.3 Header Installation

Instructions are provided to publish include files. Sources in other packages may include such files as if they were in a subdirectory with the name of the originating package. For example, the C++ header Class.hpp from package demo is included as follows:

```
#include "demo/Class.hpp"
```

All the include files in a directory (and its subdirectories) may be published by listing the directory in HEADER_DIR. All the files in the listed package subdirectory with the appropriate extension are published. If additional extensions are listed in INCLUDE_TYPES, then the corresponding files will also be published. Files in subdirectories of the header directory are made available including the directory path.

Despite the apparent convenience of such an implementation, neither HEADER_DIR nor INCLUDE_TYPES are a convention followed by DØRunII software packages. Instead, header files are usually contained in a subdirectory that has the same name as the package. Occasionally, the header files are contained in the same subdirectory as the implementation and test files. Rarely, no division into subdirectories is made and headers as well as implementation and test files are contained in the top-level package directory.

3.3.4 Library dependencies

Each package builds at most one library whose name is the same as that of the package. The library will generally depend on other libraries, *i.e.*, there are other libraries which are used by the objects contained in the package library. These libraries are listed in the top-level instruction file LIBDEPS. Whenever the package library is used for linking, all of the libraries listed in LIBDEPS are listed after it on the link line.

LIBDEPS is certainly a convention followed by almost every software package in the release. However, although this file is almost always included, for various reasons it is often constructed improperly. This file must contain every package that is in some way referenced by the objects in the package. Not doing so will produce undefined reference errors during the building of the libraries, which will cause problems later on.

3.3.5 Library build

The list of component that are to make up the package library is taken from the instruction files COMPONENTS. The appropriate implementation extensions are appended to the entries in this list to obtain the respective names of the source files. The sources are then compiled and archived in the package library.

Components should not have cyclic dependencies and should be listed in dependency order, *i.e.*, any component should not depend on those following it in the list.

Components may be distributed among multiple directories with a COMPONENTS instruction file in each. However, there is no guarantee as to the order in which the directories will be processed, so components in different directories should not depend on one another. Typically a package will have only one directory containing components. *A package which has no COMPONENTS files does not build a library.*

3.3.6 Component testing

A test file defining a main program must be provided for each component. The names of these files are obtained by appending the appropriate test extensions to the component name. If the file CTEST_DIR appears in the top-level package directory, then the test files are found in the subdirectory listed in that file. Otherwise (and typically) each test file resides in the same directory as its component source. The tests are then compiled and linked against the package library and its dependent libraries derived from LIBDEPS.

The resulting executable is run using a component test script which is obtained from one of three locations. First, the directory containing the component test is checked for a file with the same name as the component and the extension .sh. If that file is not present, then that directory is searched for the file run_component_test.sh. If neither is present, a default script is used.

The test is considered successful if the script returns zero. The default script runs the test executable and returns its return value.

The instructions for building other elements such as object components, integrated tests, binaries and scripts are also provided by CTEST, but they are not relevant to this project and will be omitted.

In short, the CTEST interface specification provides that build actions are controlled by a set of "instruction files" located in a package's top directory or subdirectories. A GNUmakefile is needed in each package subdirectory where any build action is to take place. CTEST supports two types of testing: "component tests," which apply to all library source files ("components") and which are only allowed to invoke lower-level objects and libraries, and "integrated tests," which are stand alone tests, and which are not restricted to dependence on lower level objects. Both kinds of tests are invoked by a script, which can either be a user-written one or a default script supplied by the CTEST implementation.

4 Tools acquired

4.1 UNIX

The DØRunII release used in this project resides in a machine called d0mino.fnal.gov which runs the IRIX version of the UNIX operating system developed by Silicon Graphics (SGI). Therefore, a familiarity with the UNIX operating system was crucial.

4.2 The shell programming language

The UNIX operating system can be broken down into three basic components: the scheduler, the file system, and the shell. The shell is the UNIX system's command interpreter. It is the part of the UNIX system that sits between the user and the system, forming a shell around the computer that is relatively consistent in its outward appearance. It also provides a powerful programming language, the shell programming language, which was used for the implementation of a preliminary project.

4.3 The vi editor

Vi is a screen editor that allows the user to see portions of a file on the terminal's screen and to modify characters and lines by simply typing at the current cursor position. Two screen editors are supplied with the UNIX System V Release 2.0. Of the two, vi is the more popular one, and the one used to perform editing during the initial stages of the project.

4.4 CVS

CVS is a version control system; it records the history of files using version numbers.

Every version of every file that has ever been created can of course just be saved, but this would waste an enormous amount of disk space. Instead, CVS stores all the versions of a file in a single file in a way that only stores the differences between versions.

The entire DØRunII release sits in a CVS repository. In order to make any changes to it, a copy of the different packages had to be extracted into a local release area. Once the changes had been made, they were checked back into the repository.

4.5 Python

Python was created in the early 1990s by Guido van Rossum at Stichting Mathematisch Centrum (CWI) in the Netherlands as a successor of a language called ABC. It is an interpreted, interactive, object-oriented programming language, often compared to Tcl, Perl, Scheme or Java.

It is a simple but extremely powerful programming language: it supports modules, classes, exceptions, very high level dynamic data types, and dynamic typing. It also supports interfaces to many system calls and libraries, as well as to various windowing systems (X11, Motif, Tk, Mac, MFC). The Python implementation is also portable: it runs on many brands of UNIX, on Windows, DOS, OS/2, Mac, Amiga... Because of these characteristics, Python was the language used to implement all major scripts related to the different projects.

5 My project

My summer project involved doing an analysis of each of 585 DØRunII software packages concerning the following aspects:

- Identify which packages were CTBUILD and which ones SRT. Only CTBUILD packages would be modified.
- For each CTBUILD package evaluate every component test to determine whether it is 0^{th} order, 1^{st} order, or 2^{nd} order, where
  - 0^{th} order: a main program which returns zero and does not even include its own header file,

- ❖ 1st order: a 0th order test which includes its own header file. This type of component test tests the syntactical correctness of the code in the header as well as the correctness of dependencies,
- ❖ 2nd order: a 1st order test which invokes the constructor of each class defined in the implementation file.

- Write at least a 1st order component test. Compile and run the component tests in a local release area by doing a `gmake alltest` operation.
- Write a 2nd order component test if time permits. This objective, however, was eventually dropped, since constructor calls required a deeper understanding of the implementation and header sources, and therefore made it extremely difficult to process in an automatic way.

The relevance of such a project arose from a concern put forth by a number of DØ people about the apparent lack of testing in DØRunII software packages. The estimate was that about 50% of the packages in DØRunII did not have meaningful test and thus some of the code that was being included in the release was probably not even compiling. Without appropriate testing this can happen. In the case of a 0th order test the header is not even being compiled and therefore there is no need for it to be working in order to build the release without errors. The consequence of this is that it is only when someone tries to use the code that we know it does not work.

5.1 Inventory

The very first task to be accomplished involved evaluating each package in DØRunII to determine which ones were CTBUILD and which ones were not, presumably SRT packages. The reason for this was that it was needed to assess the scope of the project. The only packages that would be altered were CTBUILD packages.

A small script was implemented in the shell programming language to do this (Appendix A). The script recognized build packages by the presence of a GNUmakefile. GNUmakefiles with "ctbuild" in them were considered CTBUILD packages; those without "ctbuild" were considered SRT packages. Those packages without a GNUmakefile altogether were included in the "other" category. The results obtained with the script follow:

- CTBUILD: **484 packages**
- SRT: **73 packages**
- Other: **28 packages**

5.2 First approach to test insertion

Once this initial task was accomplished and a proper assessment of the size of the project was made, the analysis of the first subsystem was started.

Not all subsystems are similar in terms of structure and organization. Many of them were developed with different objectives in mind and some of them offered significant difficulties for this very reason.

It was suggested to start with the Muon subsystem, which seemed to be quite well structured and as difficulty-free as could be expected.

Once the packages were extracted from the CVS repository each component test was examined looking for a statement of the form

```
#include "package_name/component_name.hpp"
```

If such a line of code was found, the component test was listed as a first order test and the next component test was then examined. However, if this line of code was not found a check for the existence of the appropriate header file was performed. If the header was found this line was added to the source of the component test.

The Muon subsystem had 196 component tests in 23 packages. Of these, 127 were 0^{th} order component tests, *i.e.*, absolutely empty tests of the form

```
main{return 0}
```

It was evident that the code was not being appropriately tested by developers. The project was already showing its value.

After these changes were made, a `gmake alltest` revealed errors in two packages due to an incomplete LIBDEPS file. The changes were checked back into CVS, a new version number was assigned to each modified package and a request for this new version to be included in the following release was made.

Vertexing subsystem, which was very similar to Muon in its structure, was then analyzed. Of the 176 component tests in its 27 packages, 55 were found to be empty.

A `gmake alltest` on this subsystem revealed a header file that was not compiling because it contained a reference to a class which was not being defined anywhere. The solution was to include the referenced header in the appropriate source file.

The project was evidently producing good results. Clearly, though, the initial approach of doing things by hand was not up to the job. For instance, as the analysis of the component tests progressed, mistakes were occasionally made in writing down the name of the component headers due to their length and complexity. This would then produce fatal errors at compilation time. In addition to this, analyzing just these two subsystems consumed an enormous amount of time: about two weeks were needed to do their analysis. To complete the remaining 24 subsystems of the DØRunII program suite would have required almost a half year!

5.3 A new approach to test insertion

I proposed to my supervisor automating component test insertion by writing a Python script to do it. The idea was attractive, but its implementation seemed at first quite challenging due to the nature of the structure of the packages in the release. Even though most packages had adopted the CTEST interface, there were many different variations within CTEST. The release was apparently not standardized enough.

A first draft of a script was written using the knowledge acquired from Muon and Vertexing. This initial draft was fairly simple. It accounted for all variations that were implemented in Muon and Vetexing.

The mechanisms for checking for the appropriate include string and for writing it in the corresponding component test were designed. It was thought these would not be significantly altered by subsequent discoveries of alternate package structures.

5.3.1 Initial script structure

This initial script looked for a COMPONENTS file in order to derive the list of components to be processed. In the case of Muon and Vertexing this file was found to appear in a directory called 'src',

along with the component tests. The script then looked for the header files in a directory with the same name as the package.

5.3.2 Subsequent observations and script modifications

As progress was being made with the analysis of other subsystems a few observations were made:

- The COMPONENTS file was found to be mainly in three different places:
  - in a directory called 'src'.
  - in the top-level package directory.
  - in other subdirectories with arbitrary names, such as 'base', 'material', 'channel', but which are not specified by any convention.
- The component tests were found to be in four different places, the first three being shared with the implementation files, *i.e.*, together with the COMPONENTS file
  - in a directory called 'src'.
  - in the top-level package directory.
  - in other subdirectories with arbitrary names, such as 'base', 'material', 'channel', but which are not specified by any convention.
  - in a directory called 'test', separated from the implementation files.
- Header files were found to be in two locations:
  - in a directory with the same name as the package.
  - in the same directory where the implementation and test files were located.

Other obscure package structures were also found but occurred rarely so they were simply not handled automatically.

With these observations I decided to develop three versions of the script that would account for all these different variations. This implementation, however, still required that the package structure be examined in advance and that the code be modified accordingly, *i.e.*, the name of the appropriate directories had to be changed where it applied.

One version accounted for the most common package structure. It looked for the COMPONENTS file and test files in 'src' and for the header files either in the same directory where test files were found or in a directory with the same name as the package.

A different version accounted for the case where the test files were found in a different directory from the COMPONENTS file, most commonly in a subdirectory called 'test'.

The last version accounted for the case where the test files and the COMPONENTS file were found in arbitrary subdirectories. This version also included an additional modification in the structure of the include string, since an additional subdirectory was needed in the path of the header file.

Although these three versions of the script made it possible to insert the tests automatically, the structure of the package still had to be examined in advance to determine which script had to be used and which changes had to be made to the code. For the purposes of finishing the project in a fast and effective way this was certainly enough, but for the purposes of leaving a tool that could be run by any developer it wasn't.

5.3.3 Project completion and final script upgrade

Using these three scripts the project was finished in just a few weeks producing extremely good results. After its completion, I decided to design one single script that would merge the three scripts

and that would require no preliminary package structure analysis. This would serve to any developer either as a reporting tool that would flag missing tests or headers files, or as a tool for actually automatically generating simple tests in every component test of a given package.

The summary of the results of adding component tests into every DØRunII CTBUILD package follows:

Total number of subsystems processed: **23**
Total number of packages processed: **495**
Total number of component tests processed: **3301**
Total number of added tests: **1559**

Total number of errors found and fixed due to the inclusion of these tests: 44

Gathering all the information acquired from the analysis of the entire release I decided that a script that would account for every possible package structure in the release without the need of a preliminary examination could be written.

5.4 Script structure

The structure of the final script is explained in detail below. For the complete code, please refer to Appendix B.

testGenerator.py

```
Arguments are package names.

def getSubdirs(packageDirectory):
        this subroutine returns the contents of SUBDIRS if it exists:
        1.  construct the path of the file SUBDIRS in the top-level package directory
        2.  if the file is not found, return 0
        3.  if the file is found
            a.  read its contents into a list
            b.  when we are done, if the list is empty, return 0
            c.  if something was appended to the list, return the list

def getComponents(componentsDirectory):
        this subroutine returns the contents of COMPONENTS if it exists:
        1.  construct the path of the file COMPONENTS in the top-level package directory
        2.  if the file is not found, return 0
        3.  if the file is found
            a.  read its contents into a list
            b.  when we are done, if the list is empty, return 0
            c.  if something was appended to the list, return the list

def analyzeComponentTests(componentsList, subdir, package, packageDirectory):
        this subroutine performs the analysis of the list of components of a given package
        1.  initialize the list that will contain components with 0th order tests and the list of
            component that will contain 1st order tests
        2.  print the list of components to standard output
        3.  assign default values to the directory containing the sources of the tests, to the
            directory containing the sources of the headers, and to the header name
        4.  assign a tuple that contains all possible locations for test sources
        5.  look for each of these locations and append to a list the ones that exist
        6.  for each component
            a.  look for the corresponding test file in each possible directory until it is found.
            b.  if the source is not found, display an error message and continue with the next
                component
            c.  if the source is found
                i.  assign a tuple that contains all possible locations for header sources
               ii.  look for each of these locations and append to a list the ones that exist
```

```
            iii.  look for the corresponding header file with every possible extension in each
                  possible directory until it is found
             iv.  if the header is not found, display an error message and continue with the
                  next component
              v.  if the header is found
                  1.  construct the appropriate include string depending on where the header
                      file was found
                  2.  look for the include string in the source of the component test
                      a.  if an illegal include string is found, display an error message and
                          append the name of the component to a list containing the names of the
                          components that already include their own header file.
                      b.  if the correct include string has been found
                          i.  if the include string has been commented out, display a warning and
                              append the name of the component to a list containing the names of
                              the components that do not include their own header file.
                         ii.  if the include string has not been commented out, append the name
                              of the component to a list containing the names of the components
                              that already include their own header file.
                      c.  if the string has not been found, append the name of the component to a
                          list containing the names of components that do not include their own
                          header file and call a subroutine generateFirstOrderTests() to insert
                          the string in the appropriate component test.

def generateFirstOrderTest(testFilesDirectory, component, includeString):
        this subroutine inserts the appropriate #include statement corresponding to the own
        component's header file
        1.  create a copy of the source file that is going to be modified.
        2.  insert the include string

def main():
        1.  check that environment variables are set. If they are not, exit.
        2.  for each package in the arguments list
            a.  construct the package directory.
            b.  if the directory is not found, display an error message and continue with the next
                package.
            c.  call the subroutine getSubdirs() to get the subdirectories that need to be
                processed
            d.  if the function call returns 0, make the components directory be the same as the
                top-level package directory.
            e.  if the function call returns a non zero value, construct the components directory
                for each directory returned by the function call.
            f.  call the subroutine analyzeComponentTests() for each components directory
            g.  print the results to standard output
```

5.5 A note on syntax

It is worth mentioning some interesting points about the code just explained.

An interesting syntactical construction was designed to surmount the problems posed by the original need to have three different versions of the script.

In order to solve this difficulty, *i.e.*, where to look for the sources, the CTEST convention was exploited. The SUBDIRS file was used to obtain a list of all directories that needed to be processed. However, this still proved not to be sufficient, since there were certain alternatives that were not accounted for by it. The 'test' directory, where many packages seemed to have their test sources, separated from the implementation sources, was not included in SUBDIRS.

A tuple containing all possible directories, such as 'test' and those listed in SUBDIRS, was created. The script then looked for each of these directories before performing any further action.

Once the actual list of available possibilities was obtained, the script looped through each of these options until the target was found, instead of creating a cascade of if-statements which would not be easy to maintain. Any new package structures could just be added to the tuple.

The same syntactical construction was implemented when looking for the header sources. In this particular case, however, it was used twice since it was necessary to implement it also for the different accepted header extensions.

6 Conclusions

The project was extremely successful. Not only was it possible to process the entire software release, but it was possible to do it in a very efficient way which produced a tool that can be used by developers as a reporting device as well as a way of testing the syntactical correctness of their sources.

The technique that was developed for ultimately approaching this problem proved that it is possible to process certain aspects of the release in an automatic way. Other immediate applications of this technique have already been proposed. This technique also proved that the features of CTEST can be fully exploited for accomplishing tasks seemingly unrelated to the original purpose of such a convention.

It is also worth mentioning that the original estimate that about 50% of the code in the release was not being tested properly proved to be correct: 47.2% of the component tests in the release did not test anything.

7 Automating DØ Code Releases Status Generation

As substantial progress was being made with the insertion of component tests into DØRunII software packages, another interesting project was suggested. This project involved writing a script that would generate an HTML web-page displaying the current status of the DØ code releases.

7.1 Background: DØ Code releases

DØ makes weekly test releases of all its software packages to IRIX and Linux. Usually, the 3 most recent test releases are available on disk, while production releases happen separately for major products, approximately every 2 to 3 months.

The test releases are the ones that are actively being developed. Essentially any developer can add changes to it. The release is built several times, each time sending email notifications to the developers responsible for the packages that break.

Due to this extremely dynamic nature, the test releases are not very reliable. Changes are stopped, however, when the test release is frozen. Approximately every three months, a production release is started based on a previous frozen test release. In the production release no development is allowed. Its purpose is to fix bugs and make minor necessary changes.

In the past, the web-page that contained the status of the different releases had had to be maintained manually but this turned out to be too time consuming.

7.2 Code releases status

In the very first stage of this project it had to be decided what information about the release would be included in the status page. After some discussions with David Ritchie and Alan Jonckheere, it was decided that the release status page would have the following structure:

- A first web-page would serve as an overview of all releases, displaying whether they were on disk or not, and also displaying the date and time that the inventory file of the release was

last modified. The latter would be a way of determining whether the release was being currently modified or not.

- A second web-page would display each release along with the current build number in each of the three machines that are used for building (d0mino.fnal.gov, d0lxbld4.fnal.gov and d0lomite.fnal.gov) and each of the two possible compiler setups (debug or maxopt)
- A third web-page would display each release along with the freeze status of that release in each of the three machines and for each format. The freeze process involves compressing the release into tar files.
- A fourth web-page would display each release along with the number of broken packages of the build specified in the Build Status Page. In case the release was currently being built, it would display a message stating so.

7.3 The script

7.3.1 HTMLgen

Apart from the mechanism for retrieving the information about the releases that had to be designed, a way of displaying this information in HTML format was also needed. For that, an already developed tool called HTMLgen was used.

HTMLgen is a Python class library for the generation of HTML documents developed by Robin Friedrich. It includes features such as the customization of document template graphics and colors through the use of resource files, minimizing the need for modifying the module source code. It also supports tables, frames, forms (persistent and otherwise) and client-side imagemaps.

The display of the information would be implemented using an instance of a class provided by HTMLgen—Series Document—for each web-page, which would then be interconnected using navigation buttons.

7.3.2 ReleaseStatus.py skeleton

Below is the class skeleton for ReleaseStatus.py. For the complete code, please refer to Appendix C.

```
class Tools:
    def exists(self, objective, path, input):

class Command:
    def __init__(self, commandName):
    def execute(self):

class Build:
    def __init__(self, hostName, version, release):
    def getCurrentBuildNumber(self, buildDir, buildList):
    def getBrokenPackages(self, buildDir, buildList):
    def getBuildData(self, resultsDir, results):

class Release:
    def __init__(self, releaseName):
    def getDateLastModified(self):
    def getBuildStatus(self):
    def getFreezeStatus(self):
    def generateAnnotationsFile(self):

class Database:
    def __init__(self, databaseName):
    def retrieveReleaseNames(self):
    def retrieveReleases(self):
    def update(self, listOfReleases):
```

```
class StatusPages:
    def __init__(self):
    def getReleaseStatus(self):
    def overviewStatusPage(self, fileName, before=None, after=None, top=None, home=None):
    def buildStatusPage(self, fileName, before=None, after=None, top=None, home=None):
    def freezeStatusPage(self, fileName, before=None, after=None, top=None, home=None):
    def errorsStatusPage(self, fileName, before=None, after=None, top=None, home=None):
    def annotationsPage(self, releaseName):
    def generate(self):
```

7.3.3 Script main features

The script has an objected oriented design in its entirety, since, in this way, other Python modules can resuse its code by just importing the appropriate attributes from it. In order to generate the release status pages an instance of the class StatusPages must be created and a call to its attribute generate() must be made.

One of the most interesting features of the script involves a mechanism for generating a persistent database of older releases. The `shelve` module provided by the Python standard library was used for this purpose. This module uses Python's database handlers to implement persistent dictionaries (similar to hash tables). A shelve object uses string keys, but the value of the object can be of any datatype as long as it can be handled by the `pickle` module.

Each release was implemented as an object that contained certain attributes, such as the release name, whether the release was still on disk or not, the implementation of the methods that retrieve the information about the release and even other Build objects.

For each release six different build objects were implemented. Each build object corresponded to the release being built in one of the three build machines (d0mino.fnal.gov, d0lxbld4.fnal.gov or d0lomite.fnal.gov,) and to each of the two different compiler setups for building (debug or maxopt.)

Once each build object was constructed, the method getBuildData() was called for each object, which would retrieve the current build number for that release and the number of broken packages as well. As it can be seen in the class skeleton shown above, the class Build contains three other methods in addition to getBuildData(), of which __init__() is just the class constructor. The other two, getCurrentBuildNumber(self, buildDir, buildList) and getBrokenPackages(self, buildDir, buildList) perform the tasks just mentioned.

7.3.4 Challenges

As it was briefly mentioned, the code releases are built in three different machines:

- d0mino.fnal.gov
- d0lxbld4.fnal.gov
- d0lomite.fnal.gov

It was then necessary to design a mechanism to send commands across the network and to get their output back in the local environment.

Python's `popen` command provided a way of interacting with the operating system from a Python script; it was still necessary to send the commands from one operating system to the other across the network.

For this, the UNIX command `rsh` was used. `rsh` connects to a host specified as the first argument and executes a command supplied to it as the second argument. It copies its standard input to the remote command, the standard output of the remote command to its standard output, and the

standard error of the remote command to its standard error. Interrupt, quit and terminate signals are propagated to the remote command.

Another difficulty arose in the implementation of the method to obtain the number of broken packages in a given release. This attribute method of the class Build simply counts the number of packages present in a summary file which is usually executed once the build is done by a script called find_broken.sh. However, if this file is not found the script find_broken.sh is immediately executed by the the release status script and the summary file is then analyzed.

The difficulty in this implementation was that the shell script find_broken.sh could only be executed by first doing a sequence of setups related to the release and the compilation characteristics, which included aliases.

An alias is an alternative and usually easier-to-understand name for a defined data object. This data object can be defined once and later a programmer can define one or more equivalent aliases that will also refer to that data object. Unfortunately, aliases cannot be sent with the network command `rsh`. That is, we were not able to execute find_broken.sh remotely.

A solution for this problem was eventually derived from a suggestion by Paul Russo to implement "Trojan horse" scripts for certain aspects of the release status script. Such a script can be installed in a remote computer and the only command that is sent across the network is its execution command. In this way, all the setups and a call to find_broken.sh were put together in a shell script that could be executed remotely (Appendix E).

7.3.5 Other features

A few other features were later implemented. One of them consisted in making each of the fields indicating the number of broken packages in the Error Status Page a link to the log file corresponding to that release build.

Another feature consisted in implementing a mechanism for release managers to include comments about the releases. This was implemented by writing a small script that could be run on the command line and, by providing a release name and either a string or a file to it, it would generate an HTML page with this information. These pages could be reached by a link next to the release name in any of the status pages (Appendix D).

For the generated web-pages please visit:

http://www-d0.fnal.gov/computing/releases/ReleaseStatus/html/overview.html.

8 Acknowledgments

I would like to thank greatly the three people that played an essential role in making my summer a success and an extraordinary learning experience: Alan Jonckheere, Paul Russo and my supervisor, David Ritchie. It has been a pleasure working with and learning from them.

I would also like to thank the SIST committee for giving me this opportunity as well as Dianne Engram, Dr. Davenport, Elliott McCrory, my mentor Cosmore Sylvester, my fellow interns and all the people that in any way helped for making this summer a remarkable experience.

9 References

"Physics Highlights from the DØ Experiment 1992-1999." http://d0server1.fnal.gov/projects/results/runi/highlights/runi_summary.html

Adams, David L. "CTEST 2.00: an interface for building C++ packages." Rice University, 1999.

Brooijmans, G., Greenlee, H., Melanson, H., Womersley, J. "CTEST/CTBUILD Review Report" 1999.

Kochan, Stephen G. and Wood, Patrick H. Wood. Exploring the UNIX system. Hayden Book Company. New Jersey, 1984.

Lundh, Fredrik. Python Standard Library. O'Reilly & Associates, Inc. California, 2001.

Lutz, Mark and Ascher, David. Learning Python. O'Reilly & Associates, Inc. California, 1999.

Appendix A

ctbuildDB

```
#
# Determines which packages are ctbuild
# in a release specified as the first argument
# Results are stored in three files, one containing
# the list of packages that are ctbuild called
# ctbpkg, another one called notctbpkg
# that contains the list of non ctbuild
# packages and one file called others that contains
# the packages without a GNUmakefile
#

# create the output files
cat > ctbpkg
cat > notctbpkg
cat > other

# change working directory to the one specified
cd $1
packages=*

for package in $packages
do
        tmp=‘ls $package | grep GNUmakefile‘
        if [ -n "$tmp" ]
        then
                output=‘grep ctbuild $package/GNUmakefile‘
                if [ -n "$output" ]
                then
                        cd
                        echo "$package" >> ctbpkg
                else
                        cd
                        echo "$package" >> notctbpkg
                fi
                cd $1
        else
                cd
                echo "$package" >> other
        fi
        cd $1
done

# return to the working directory
cd

# display the results
echo Results
echo ctbuild packages ‘wc -l ctbpkg‘
echo not ctbuild packages ‘wc -l notctbpkg‘
echo other ‘wc -l other‘
```

Appendix B

testGenerator.py

```
#! usr/bin/env python

""" testGenerator.py -- Build tool.
Evaluate each component test of a given package to determine whether it is zeroth order
or first order, where zeroth order is a component test that has a main function but
without including its own include file and first order is a component test that
includes its own include file.

Generate first order tests for the components having only a zeroth order test.

The program should be executed after having set the variable SRT_PRIVATE_CONTEXT
to the local release area, and by passing to it the package name as an argument."""

import os
import re
import string
import sys

packagesList = []

if len(sys.argv) <= 1:
        print '\nERROR: No package names provided.'
        sys.exit(0)
else:
        for argv in sys.argv[1:]:
                if argv[-1] == '/':
                        packagesList.append(argv[:-1])
                else:
                        packagesList.append(argv)


def getSubdirs(packageDirectory):
        subdirs = os.path.join(packageDirectory, 'SUBDIRS')
        if not os.path.isfile(subdirs):
                print '\nWARNING: Could not find %s.' % subdirs
                return 0

        subdirsList = []
        f = open(subdirs, 'r')
        for line in f.xreadlines():
                if re.search('(|\s+)#(|\s+)\S+', line): continue
                m = re.search('\w+',line)
                if m: subdirsList.append(m.group(0))
        f.close()
        if len(subdirsList) == 0:
                print '\nWARNING: %s is empty.' % subdirs
                return 0
        subdirsList.sort()
        return subdirsList

def getComponents(componentsDirectory):
        components = os.path.join(componentsDirectory, 'COMPONENTS')
        if not os.path.isfile(components):
                print '\nCould not find %s.' % components
                return 0

        componentsList = []
        f = open(components, 'r')
        for line in f.xreadlines():
                if re.search('(|\s+)#(|\s+)\S+', line): continue
                m = re.search('\w+', line)
                if m: componentsList.append(m.group(0))
        f.close()
        if len(componentsList) == 0:
                print '\nWARNING: %s is empty.' % components
```

```
                return 0
        componentsList.sort()
        return componentsList

def analyzeComponentTests(componentsList, subdir, package, packageDirectory):
        zerothOrderTests = []
        firstOrderTests = []

        # print the list of components
        print '\nComponent list in %s directory:\n' % subdir
        for component in componentsList:
                print component + ',',
        print '\n'

        # default assignments
        testDir = subdir
        testDirectory = os.path.join(packageDirectory, testDir)
        componentHeader = component + '.hpp'
        headerDir = package

        testDirectories = []
        testDirsTuple = (subdir, 'test')
        for testDir in testDirsTuple:
                testDirPath = os.path.join(packageDirectory, testDir)
                if os.path.isdir(testDirPath):
                        testDirectories.append(testDir)

        for component in componentsList:
                # check that a test for this component exists
                componentTestName = component + '_t.cpp'
                for testDir in testDirectories:
                        testDirectory = os.path.join(packageDirectory, testDir)
                        if componentTestName in os.listdir(testDirectory):
                                componentTest = os.path.join(testDirectory, componentTestName)
                                break
                else:
                        print 'ERROR: Could not find a component test for %s.' % component
                        continue

                headerDirectories = []
                headerDirsTuple = (package, subdir)
                for headerDir in headerDirsTuple:
                        headerDirPath = os.path.join(packageDirectory, headerDir)
                        if os.path.isdir(headerDirPath):
                                headerDirectories.append(headerDir)

                # check that a header file for this component exists
                headerExtensions = ('.hpp', '.h', '.hh', '.H')
                for headerDir in headerDirectories:
                        headerDirectory = os.path.join(packageDirectory, headerDir)
                        headerDirectoryList = os.listdir(headerDirectory)
                        headerFound = 0
                        for extension in headerExtensions:
                                componentHeader = component + extension
                                if componentHeader in headerDirectoryList:
                                        headerFound = 1
                                        break
                        if headerFound: break
                else:
                        print 'ERROR: Could not find a component header for %s.' % component
                        continue

                includeString1 = '%s/%s' % (package, componentHeader)
                includeString2 = componentHeader

                if headerDir == package or headerDir == 'src':
                        includeString1 = '%s/%s' % (package, componentHeader)
                        includeString2 = componentHeader
                        includeStringPattern1 = re.compile('#include\s+("%s"|<%s)'
                                                           %(includeString1, includeString1))
                        includeStringPattern2 = re.compile('#include\s+("%s"|<%s>)'
```

```
                                                % (includeString2, includeString2))
                else:
                        includeString1 = '%s/%s/%s' % (package, headerDir, componentHeader)
                        includeString2 = componentHeader
                        includeStringPattern1 = re.compile('#include\s+("%s"|<%s>)'
                                                % (includeString1, includeString1))
                        includeStringPattern2 = re.compile('#include\s+("%s"|<%s>)'
                                                % (includeString2, includeString2))

                f = open(componentTest, 'r')
                stringFound = 0
                for line in f.xreadlines():
                        if headerDir == testDir:
                                if includeStringPattern1.search(line):
                                        includeStringPattern = includeStringPattern1
                                elif includeStringPattern2.search(line):
                                        includeStringPattern = includeStringPattern2
                                        print 'WARNING: Illegal include statement found in' +
                                                component test %s.' % componentTestName
                                else: continue
                        else:
                                includeStringPattern = includeStringPattern1
                                if not includeStringPattern.search(line): continue

                        # check that the string has not been commented out
                        if includeStringPattern == includeStringPattern1:
                                commentedLinePattern = re.compile('//(|s+)#include\s+("%s"|<%s>)'
                                                % (includeString1, includeString1))
                        else:
                                commentedLinePattern = re.compile('//(|s+)#include\s+("%s"|<%s>)'
                                                % (includeString2, includeString2))
                        if commentedLinePattern.search(line):
                                print '<--> WARNING: The include statement of the component test' +
                                      ' ' + componentTestName + ' has been commented out. A new ' +
                                      'uncommented include string will be added.'
                                continue
                        else:
                                stringFound = 1
                                break
                f.close()

                if stringFound:
                        print 'Component test %s already includes its include file.'
                                                                % componentTestName
                        firstOrderTests.append(componentTestName)
                else:
                        print 'Component test %s does not include its include file.'
                                                                % componentTestName
                        zerothOrderTests.append(componentTestName)
                        generateFirstOrderTest(testDirectory, component, includeString1)
        return(zerothOrderTests, firstOrderTests)

def generateFirstOrderTest(testFilesDirectory, component, includeString):
        componentIncludeString = '#include "%s"' %(includeString)
        print '<--> Adding component include string \'%s\'...' % componentIncludeString,

        componentTestName = component + '_t.cpp'
        componentTest = os.path.join(testFilesDirectory, componentTestName)
        backupFileName = component + '_t_original.cpp'
        backupFile = os.path.join(testFilesDirectory, backupFileName)
        os.rename(componentTest, backupFile)

        input = open(backupFile, 'r')
        output = open(componentTest, 'w')
        includeStringWrittenToFile = 0
        for line in input.xreadlines():
                if line[0:2] == '//': output.write(line)
                elif not includeStringWrittenToFile:
                        output.write('\n' + componentIncludeString + '\n\n')
                        includeStringWrittenToFile = 1
                        if line[0] == '\n': continue
```

```
                else: output.write(line)
            else: output.write(line)
        input.close()
        output.close()
        print 'done.'


def main():
        publicContext = os.environ.get('SRT_PUBLIC_CONTEXT')
        if not publicContext:
                print '\nERROR: SRT_PUBLIC_CONTEXT is not set.'
                sys.exit(127)

        privateContext = os.environ.get('SRT_PRIVATE_CONTEXT')
        if not privateContext:
                print '\nERROR: SRT_PRIVATE_CONTEXT is not set.'
                sys.exit(127)

        for package in packagesList:
                packageDirectory = os.path.join(privateContext, package)
                if not os.path.isdir(packageDirectory):
                        print '\nERROR: Could not find %s.' % packageDirectory
                        continue

                # print summary header
                print '\n#\n#-----<%s>-----\n#\n' % package

                subdirsList = getSubdirs(packageDirectory)
                if subdirsList == 0:
                        # look for a components file in the top-level of the package structure
                        componentsDirectory = os.path.join(packageDirectory)
                        componentsList = getComponents(componentsDirectory)
                        if componentsList == 0: continue
                        zerothOrderTests, firstOrderTests = analyzeComponentTests(componentsList,
                                                                subdir, package, packageDirectory)

                        # report results
                        print '\nPackage %s summary.\n' % package
                        print 'Zeroth order component tests:'
                        for component in zerothOrderTests:
                                print '\t%s' % component
                        print '\nFirst order components tests:'
                        for component in firstOrderTests:
                                print '\t%s' % component
                        print '\nTotal number of components: %d.' % len(componentsList)
                        print 'Total number of modified tests: %d.' % len(zerothOrderTests)
                else:
                        # look for a components file in each subdirectory
                        for subdir in subdirsList:
                                componentsDirectory = os.path.join(packageDirectory, subdir)
                                if not os.path.isdir(componentsDirectory):
                                        print '\nERROR: Could not find %s.' % componentsDirectory
                                        continue

                                componentsList = getComponents(componentsDirectory)
                                if componentsList == 0: continue
                                zerothOrderTests, firstOrderTests =
                                                        analyzeComponentTests(componentsList,  subdir,
                                                                        package, packageDirectory)

                                # report results
                                print '\nPackage %s summary.\n' % package
                                print 'Zeroth order component tests:'
                                for component in zerothOrderTests:
                                        print '\t%s' % component
                                print '\nFirst order components tests:'
                                for component in firstOrderTests:
                                        print '\t%s' % component
                                print '\nTotal number of components: %d.' % len(componentsList)
                                print 'Total number of modified tests: %d.' % len(zerothOrderTests)
```

```
if __name__ == '__main__':
        # We are being run as a script.
        main()
        sys.exit(0)
else:
        # We are being loaded as a module.
        pass
```

Appendix C

ReleaseStatus.py

```
#! /usr/bin/env python

"""ReleaseStatus.py generates four release status pages."""

#
# Module imports.
#

import commands
import glob
import os
import os.path
import re
import shelve
import stat
import string
import sys
import time
import HTMLgen.barchart as barchart
from HTMLgen.HTMLcolors import *
from HTMLgen.HTMLgen import *

#
# Global data specifications.
#

__author__ = 'Mariano Zimmler zimmler@fnal.gov'
__version__ = '0.4.0'

releaseStatusGen = os.path.split(sys.argv[0])[1]
releaseStatusDir = sys.path[0]

imageDir = os.path.join('..', 'image')
dataDir = os.path.abspath(os.path.join(releaseStatusDir, 'data'))
htmlDir = os.path.abspath(os.path.join(releaseStatusDir, 'html'))
annotationsDir = os.path.abspath(os.path.join(releaseStatusDir, 'annotations'))
docTitle = 'Release Status Page (Version: %s)' % __version__

while not os.path.exists(os.path.abspath(os.path.join(releaseStatusDir, releaseStatusGen))):
    print "Program ", releaseStatusGen, " not found within ", releaseStatusDir, " directory."
    releaseStatusDir = raw_input("Specify directory path to " + releaseStatusGen + " or press
                                                                    only ENTER to exit. ")
    if releaseStatusDir == "": sys.exit(1)
    print "Directory path ", releaseStatusDir, " will be inserted into sys.path[0]"
    sys.path.insert(0,releaseStatusDir)
    releaseStatusDir = sys.path[0]

try:
    os.mkdir(htmlDir)
    print "(ReleaseStatus.py) - made html sub-directory"
except os.error:
    pass
    print "(ReleaseStatus.py) - did not make html sub-directory (may already exist)."

try:
    os.mkdir(annotationsDir)
    print "(ReleaseStatus.py) - made annotations sub-directory"
except os.error:
    pass
    print "(ReleaseStatus.py) - did not make annotations sub-directory (may already exist)."

releasesDir = '/d0dist/dist/releases'
scriptsDir = '/RunII/home/d0relmgr'

d0mino = 'd0mino.fnal.gov'
```

```
d0lxbld4 = 'd0lxbld4.fnal.gov'
d0lomite = 'd0lomite.fnal.gov'

#
# Class definitions.
#

class Tools:
    def exists(self, objective, path, input):
        m = re.compile(objective)
        for line in input:
            if m.search(line): return line
            else: continue
        else:
            print 'WARNING: Could not find %s.' % path
            return 0

class Command:
    def __init__(self, commandName):
        self.name = commandName

    def execute(self):
        f = os.popen(self.name)
        output = f.readlines()
        f.close()
        return output

class Build:
    def __init__(self, hostName, version, release):
        self.host = hostName
        self.platform = version
        self.releaseName = release
        self.buildNumber = '-'
        self.buildNumberC = BLACK
        self.buildInProgress = 0
        self.brokenPackages = '-'
        self.linkToLog = '-'

    def getCurrentBuildNumber(self, buildDir, buildList):
        buildNumber = '-'
        buildNumberFile = os.path.join(buildDir, '.buildnum')
        path = Tools()
        if not path.exists('.buildnum', buildNumberFile, buildList):
            buildNumber = '-'
            buildNumberC = BLACK
            buildInProgress = 0
            return (buildNumber, buildNumberC, buildInProgress)

        readFile = Command("rsh -n %s -l d0relmgr -F cat %s" % (self.host, buildNumberFile))
        buildData = readFile.execute()
        for line in buildData:
            m = re.search('\d+', line)
            if m:
                buildNumber = int(m.group(0)) - 1
                break
        else:
            buildNumber = '-'
            buildNumberC = BLACK
            buildInProgress = 0
            return (buildNumber, buildNumberC, buildInProgress)

        buildInProgress = 0;
        while buildNumber > 0:
            begin = 'build-%s-0%s.begin' % (self.platform, buildNumber)
            end = 'build-%s-0%s.end' % (self.platform, buildNumber)
            if re.match('\d\d', repr(buildNumber)):
                begin = 'build-%s-%s.begin' % (self.platform, buildNumber)
                end = 'build-%s-%s.end' % (self.platform, buildNumber)
            beginPattern = re.compile(begin)
            endPattern = re.compile(end)
            for i in buildList:
```

```
            if beginPattern.search(i):
                for j in buildList:
                    m = endPattern.search(j)
                    if m: break
                else: buildInProgress = 1
                break
        else:
            buildNumber = buildNumber - 1
            continue
        break
    else: buildNumber = '-'

    if buildNumber == '-': buildNumberC = BLACK
    elif buildInProgress: buildNumberC = RED
    else: buildNumberC = GREEN6
    return (buildNumber, buildNumberC, buildInProgress)

def getBrokenPackages(self, buildDir, buildList):
    brokenPackages = []

    end = 'build-%s-0%s.end' % (self.platform, self.buildNumber)
    summary = 'summary-0%s.txt' % self.buildNumber
    if len(repr(self.buildNumber)) == 2:
        end = 'build-%s-%s.end' % (self.platform, self.buildNumber)
        summary = 'summary-%s.txt' % self.buildNumber
    endPattern = re.compile(end)
    summaryPattern = re.compile(summary)

    summaryFile = os.path.join(buildDir, summary)
    path = Tools()
    if not path.exists(summary, summaryFile, buildList):
        if re.search('-maxopt', self.platform):
            findBroken = Command("rsh -n " + self.host + " -l d0relmgr -F " +
                                 "'source " + scriptsDir + "/update_relstat_err.sh " +
                                 self.releaseName + " maxopt'")
            output = findBroken.execute()
        else:
            findBroken = Command("rsh -n " + self.host + " -l d0relmgr -F " +
                                 "'source " + scriptsDir + "/update_relstat_err.sh " +
                                 self.releaseName + " default'")
            output = findBroken.execute()
    else:
        foundEnd = 0
        foundSummary = 0
        prematureSummary = 0
        for i in buildList:
            if summaryPattern.search(i):
                foundSummary = 1
            if endPattern.search(i):
                foundEnd = 1
            if foundEnd and not foundSummary:
                prematureSummary = 1
                break
        if prematureSummary:
            if re.search('-maxopt', self.platform):
                findBroken = Command("rsh -n " + self.host + " -l d0relmgr -F"+
                                     " 'source " + scriptsDir + "/update_relstat_err.sh " +
                                     self.releaseName + " maxopt'")
                output = findBroken.execute()
            else:
                findBroken = Command("rsh -n " + self.host + " -l d0relmgr -F"+
                                     "'source " + scriptsDir + "/update_relstat_err.sh " +
                                     self.releaseName + " default'")
                output = findBroken.execute()

    readFile = Command("rsh -n %s -l d0relmgr -F cat %s" % (self.host,summaryFile))
    errorsData = readFile.execute()
    if len(errorsData) == 0:
        brokenPackages = '-'
    else:
        for line in errorsData:
```

```
                m = re.match('\w+', line)
                if m: brokenPackages.append(m.group(0))
        if brokenPackages == '-': return brokenPackages
        else: return len(brokenPackages)

    def getBuildData(self, resultsDir, results):
        buildDir = os.path.join(resultsDir, self.platform)

        path = Tools()
        if not path.exists(self.platform, buildDir, results):
            self.buildNumber = '-'
            self.buildNumberC = BLACK
            self.brokenPackages = '-'
            return None

        listDir = Command("rsh -n %s -l d0relmgr -F ls -alt %s" % (self.host, buildDir))
        buildList = listDir.execute()
        self.buildNumber, self.buildNumberC, self.buildInProgress =
                                            self.getCurrentBuildNumber(buildDir, buildList)

        if self.buildInProgress: self.brokenPackages = 'In progress'
        else: self.brokenPackages = self.getBrokenPackages(buildDir, buildList)
        if self.brokenPackages == '-': self.linkToLog = '-'
        elif self.brokenPackages == '<STRONG>In progress</STRONG>':
            self.linkToLog = '<STRONG>In progress</STRONG>'
        else: self.linkToLog = '<A HREF="http://d0ntwg02.fnal.gov/buildlogs/viewBuild.aspx?' +
                               'release=%s&platform=%s"' % (self.releaseName, self.platform) +
                               '>' + 'self.brokenPackages' + '</A>'

class Release:
    def __init__(self, releaseName):
        self.name = releaseName
        self.isOnDisk = 'yes'
        self.isOnDiskC = BLUE

    def getDateLastModified(self):
        releaseInventory = os.path.join(releasesDir, self.name, 'D0reldb', 'inventory')
        try:
            releaseInventoryStat = os.stat(releaseInventory)
            self.dateLastModified = time.ctime(releaseInventoryStat[8])
        except OSError:
            self.dateLastModified = '-'

    def getBuildStatus(self):
        releaseDir = os.path.join(releasesDir, self.name)
        resultsDir = os.path.join(releaseDir, 'results')

        self.IRIXdebug = Build(d0mino, '-', self.name)
        self.IRIXmaxopt = Build(d0mino, '-', self.name)
        self.RH6debug = Build(d0lxbld4, '-', self.name)
        self.RH6maxopt = Build(d0lxbld4, '-', self.name)
        self.RH7debug = Build(d0lomite, '-', self.name)
        self.RH7maxopt = Build(d0lomite, '-', self.name)

        # d0mino IRIX
        listDir = Command("rsh -n %s -l d0relmgr -F ls %s" % (d0mino, releaseDir))
        dirList = listDir.execute()
        path = Tools()
        if path.exists('results', resultsDir, dirList):
            listDir = Command("rsh -n %s -l d0relmgr -F ls %s" % (d0mino, resultsDir))
            results = listDir.execute()

            m = re.compile('(IRIX\d\.\d-\w\w\w_\d_\d|IRIX\d-\w\w\w_\d_\d)')
            for line in results:
                if m.search(line):
                    self.IRIXdebug.platform = m.search(line).group(0)
                    self.IRIXdebug.getBuildData(resultsDir, results)
                else: continue

            m = re.compile('(IRIX\d\.\d-\w\w\w_\d_\d-maxopt|IRIX\d-\w\w\w_\d_\d-maxopt)')
            for line in results:
```

```
            if m.search(line):
                self.IRIXmaxopt.platform = m.search(line).group(0)
                self.IRIXmaxopt.getBuildData(resultsDir, results)
            else: continue

        # d0lxbld4 Red Hat 6 / Linux 2.2 Server
        listDir = Command("rsh -n %s -l d0relmgr -F ls %s" % (d0lxbld4, releaseDir))
        dirList = listDir.execute()
        if path.exists('results', resultsDir, dirList):
            listDir = Command("rsh -n %s -l d0relmgr -F ls %s" % (d0lxbld4, resultsDir))
            results = listDir.execute()

            m = re.compile('(Linux\d\.\d-\w\w\w_\d_\d|Linux\d-\w\w\w_\d_\d)')
            for line in results:
                if m.search(line):
                    self.RH6debug.platform = m.search(line).group(0)
                    self.RH6debug.getBuildData(resultsDir, results)
                else: continue

            m = re.compile('(Linux\d\.\d-\w\w\w_\d_\d-maxopt|Linux\d-\w\w\w_\d_\d-maxopt)')
            for line in results:
                if m.search(line):
                    self.RH6maxopt.platform = m.search(line).group(0)
                    self.RH6maxopt.getBuildData(resultsDir, results)
                else: continue

        # d0lomite Red Hat 7 / Linux 2.4 Server
        listDir = Command("rsh -n %s -l d0relmgr -F ls %s" % (d0lomite, releaseDir))
        dirList = listDir.execute()
        if path.exists('results', resultsDir, dirList):
            listDir = Command("rsh -n %s -l d0relmgr -F ls %s" % (d0lomite, resultsDir))
            results = listDir.execute()

            m = re.compile('(Linux\d\.\d-\w\w\w_\d_\d|Linux\d-\w\w\w_\d_\d)')
            for line in results:
                if m.search(line):
                    self.RH7debug.platform = m.search(line).group(0)
                    self.RH7debug.getBuildData(resultsDir, results)
                else: continue

            m = re.compile('(Linux\d\.\d-\w\w\w_\d_\d-maxopt|Linux\d-\w\w\w_\d_\d-maxopt)')
            for line in results:
                if m.search(line):
                    self.RH7maxopt.platform = m.search(line).group(0)
                    self.RH7maxopt.getBuildData(resultsDir, results)
                else: continue

    def getFreezeStatus(self):
        self.IRIXdebugfreeze = 'no'; self.IRIXdebugfreezeC = RED
        self.IRIXmaxoptfreeze = 'no'; self.IRIXmaxoptfreezeC = RED
        self.RH6debugfreeze = 'no'; self.RH6debugfreezeC = RED
        self.RH6maxoptfreeze = 'no'; self.RH6maxoptfreezeC = RED
        self.RH7debugfreeze = 'no'; self.RH7debugfreezeC = RED
        self.RH7maxoptfreeze = 'no'; self.RH7maxoptfreezeC = RED

        genericFreezeCommand = 'upd list -h www-d0.fnal.gov -aK+ D0RunII'
        freezeAvailability = genericFreezeCommand + ' %s -qdist' % self.name
        upd = Command(freezeAvailability)
        commandOutput = upd.execute()
        for line in commandOutput:
            m = re.match('"D0RunII" "%s" "NULL" "dist" ""' % self.name, line)
            if m:
                self.freezeAvailable = 1
                break
        else:
            self.freezeAvailable = 0
            return None

        IRIXdebug = re.compile(r'(IRIX6\.5-KCC_4_0|IRIX6-KCC_3_4)')
        IRIXmaxopt = re.compile(r'(IRIX6\.5-KCC_4_0-maxopt|IRIX6-KCC_3_4-maxopt)')
        RH6debug = re.compile(r'(Linux2\.2-KCC_4_0|Linux2-KCC_3_4)')
```

```
        RH6maxopt = re.compile(r'(Linux2\.2-KCC_4_0-maxopt|Linux2-KCC_3_4-maxopt)')
        RH7debug = re.compile(r'(Linux2\.4-KCC_4_0|Linux2-KCC_4_0)')
        RH7maxopt = re.compile(r'(Linux2\.4-KCC_4_0-maxopt|Linux2-KCC_4_0-maxopt)')

        platformSpecificAvailability = genericFreezeCommand + '-bin %s' % self.name
        upd = Command(platformSpecificAvailability)
        commandOutput = upd.execute()
        for line in commandOutput:
            if IRIXmaxopt.search(line):
                self.IRIXmaxoptfreeze = 'yes'; self.IRIXmaxoptfreezeC = GREEN6
            elif IRIXdebug.search(line):
                self.IRIXdebugfreeze = 'yes'; self.IRIXdebugfreezeC = GREEN6
            elif RH6maxopt.search(line):
                self.RH6maxoptfreeze = 'yes'; self.RH6maxoptfreezeC = GREEN6
            elif RH6debug.search(line):
                self.RH6debugfreeze = 'yes'; self.RH6debugfreezeC = GREEN6
            elif RH7maxopt.search('line'):
                self.RH7maxoptfreeze = 'yes'; self.RH7maxoptfreezeC = GREEN6
            elif RH7debug.search(line):
                self.RH7debugfreeze = 'yes'; self.RH7debugfreezeC = GREEN6

    def generateAnnotationsFile(self):
        annotationsFileName = self.name + '.txt'
        annotationsFile = os.path.join(annotationsDir, annotationsFileName)
        emptyFile = os.path.join(releaseStatusDir, 'emptyFile')
        if not os.path.isfile(annotationsFile):
            createAnnotationFile = Command("cat %s > %s" % (emptyFile, annotationsFile))
            output = createAnnotationFile.execute()

class Database:
    def __init__(self, databaseName):
        self.name = databaseName

    def retrieveReleaseNames(self):
        releaseNames = []
        releaseDatabase = shelve.open(self.name, 'r')
        return releaseDatabase.keys()

    def retrieveReleases(self):
        releases = []
        releaseDatabase = shelve.open(self.name, 'r')
        for key in releaseDatabase.keys():
            releases.append(releaseDatabase[key])
        return releases

    def update(self, listOfReleases):
        releaseDatabase = shelve.open(self.name, 'c')
        for release in listOfReleases:
            releaseDatabase[release.name] = release
        releaseDatabase.close()

class StatusPages:
    def __init__(self):
        self.releases = []

    def getReleaseStatus(self):
        currentReleaseNames = []
        databaseReleaseNames = []
        databaseReleases = []
        releaseNames = []

        listOfEntries = os.listdir(releasesDir)

        tReleasePattern1 = re.compile(r'^t\d\d\.\d\d\.\d\d')
        tReleasePattern2 = re.compile(r'^t\d\d\d-gcc')
        pReleasePattern = re.compile(r'^p\d\d\.\d\d\.\d\d')
        oReleasePattern = re.compile(r'^onl\d\d\.\d\d\.\d\d')
        for entry in listOfEntries:
            if not tReleasePattern1.match(entry) and not tReleasePattern2.match(entry) and \
               not pReleasePattern.match(entry) and not oReleasePattern.match(entry): continue
            currentReleaseNames.append(entry)
```

```
    database = Database('releaseDatabase')
    if os.path.isfile(os.path.join(os.curdir, database.name)):
        databaseReleaseNames = database.retrieveReleaseNames()
        databaseReleases = database.retrieveReleases()

    releaseNames = currentReleaseNames[:]

    for dbReleaseName in databaseReleaseNames:
        if dbReleaseName in currentReleaseNames: continue
        releaseNames.append(dbReleaseName)

    releaseNames.sort()
    releaseNames.reverse()

    for releaseName in releaseNames:
        if releaseName in currentReleaseNames:
            release = Release(releaseName)
            release.getDateLastModified()
            release.getBuildStatus()
            release.getFreezeStatus()
            release.generateAnnotationsFile()
            self.releases.append(release)
        else:
            for dbRelease in databaseReleases:
                if dbRelease.name == releaseName:
                    print '%s retrieved from database' % releaseName
                    dbRelease.isOnDisk = 'no'
                    dbRelease.isOnDiskC = 'FF3300'
                    self.releases.append(dbRelease)

    database.update(self.releases)

def overviewStatusPage(self, fileName, before=None, after=None, top=None, home=None):
    doc = SeriesDocument('RelStat.rc')

    doc.title = docTitle
    doc.subtitle = 'Status Overview'
    doc.banner = None
    doc.logo = (os.path.join(imageDir, 'd0logo.gif'),111,68)

    doc.prev = (os.path.join(imageDir, 'BTN_PrevPage.gif'),74,19)
    doc.next = (os.path.join(imageDir, 'BTN_NextPage.gif'),74,19)
    doc.top = (os.path.join(imageDir, 'BTN_ManualTop.gif'),74,19)
    doc.home = (os.path.join(imageDir, 'BTN_HomePage.gif'),74,19)
    doc.goprev = before
    doc.gonext = after
    doc.gotop = top
    doc.gohome = home

    rowContents = []
    for release in self.releases:
        line = ['<STRONG>' + release.name + ' </STRONG><A HREF="%s.html"' % release.name +
                                                                        '>[A]</A>',
                Bold(Font(release.isOnDisk, color=release.isOnDiskC)),
                Bold(release.dateLastModified)]
        rowContents.append(line)

    relTable = Table('Releases On Disk',
        heading=['Release','On Disk','<TT>D0reldb/inventory</TT><BR> Date Last Modified'],
        body_color=['#DDDDDD','#FFFFFF','#FFFFFF'],
        heading_color=['#DDDDDD','#FFFFFF','#FFFFFF'],
        heading_align='center',
        column1_align='center',
        cell_align='center',
        body=rowContents)
    doc.append(relTable)

    doc.append(HR())
    doc.append_file(os.path.join(dataDir, 'info-txt.html'))
    doc.write(os.path.join(htmlDir, fileName))
```

```
    def buildStatusPage(self, fileName, before=None, after=None, top=None, home=None):
        doc = SeriesDocument('RelStat.rc')
        doc.title = docTitle
        doc.subtitle = 'Build Status'
        doc.banner = None
        doc.logo = (os.path.join(imageDir, 'd0logo.gif'),111,68)

        doc.prev = (os.path.join(imageDir, 'BTN_PrevPage.gif'),74,19)
        doc.next = (os.path.join(imageDir, 'BTN_NextPage.gif'),74,19)
        doc.top = (os.path.join(imageDir, 'BTN_ManualTop.gif'),74,19)
        doc.home = (os.path.join(imageDir, 'BTN_HomePage.gif'),74,19)
        doc.goprev = before
        doc.gonext = after
        doc.gotop = top
        doc.gohome = home

        rowContents = []
        for release in self.releases:
            line = ['<STRONG>' + release.name + ' </STRONG><A HREF="%s.html"' % release.name +
                                                                                '>[A]</A>',
                    Bold(Font(release.isOnDisk, color=release.isOnDiskC)),
                    Bold(Font(release.IRIXdebug.buildNumber,
                                                color=release.IRIXdebug.buildNumberC)),
                    Bold(Font(release.IRIXmaxopt.buildNumber,
                                                color=release.IRIXmaxopt.buildNumberC)),
                    Bold(Font(release.RH6debug.buildNumber,  color=release.RH6debug.buildNumberC)),
                    Bold(Font(release.RH6maxopt.buildNumber,
                                                color=release.RH6maxopt.buildNumberC)),
                    Bold(Font(release.RH7debug.buildNumber,  color=release.RH7debug.buildNumberC)),
                    Bold(Font(release.RH7maxopt.buildNumber,
                                                color=release.RH7maxopt.buildNumberC))]
            rowContents.append(line)

        relTable = Table('Current Release Build',
            heading=['Release','On Disk','IRIX','IRIX<BR>maxopt','Red Hat 6','Red Hat ' +
                     '6<BR>maxopt','Red Hat 7','Red Hat 7<BR>maxopt'],

body_color=['#888888','#DDDDDD','#CCCCCC','#BBBBBB','#CCCCCC','#BBBBBB','#CCCCCC','#BBBBBB'],

heading_color=['#888888','#DDDDDD','#CCCCCC','#BBBBBB','#CCCCCC','#BBBBBB','#CCCCCC','#BBBBBB'],
            heading_align='center',
            column1_align='center',
            cell_align='center',
            body=rowContents)
        doc.append(relTable)

        doc.append(HR())
        doc.append('The following provides the proper interpretation of the information above.')
        ilist = [(Image(os.path.join(imageDir, 'green_dot.gif')), Bold('The build is finished.')),
                 (Image(os.path.join(imageDir, 'red_dot.gif')), Bold('The build is in
                                                                                progress.'))]
        doc.append(Heading(4, 'Legend'))
        doc.append(ImageBulletList(ilist))

        doc.append(HR())
        doc.append_file(os.path.join(dataDir, 'info-txt.html'))
        doc.write(os.path.join(htmlDir, fileName))

    def freezeStatusPage(self, fileName, before=None, after=None, top=None, home=None):
        doc = SeriesDocument('RelStat.rc')
        doc.title = docTitle
        doc.subtitle = 'Freeze Status'
        doc.banner = None
        doc.logo = (os.path.join(imageDir, 'd0logo.gif'),111,68)

        doc.prev = (os.path.join(imageDir, 'BTN_PrevPage.gif'),74,19)
        doc.next = (os.path.join(imageDir, 'BTN_NextPage.gif'),74,19)
        doc.top = (os.path.join(imageDir, 'BTN_ManualTop.gif'),74,19)
        doc.home = (os.path.join(imageDir, 'BTN_HomePage.gif'),74,19)
        doc.goprev = before
```

```
        doc.gonext = after
        doc.gotop = top
        doc.gohome = home

        rowContents = []
        for release in self.releases:
            line = ['<STRONG>' + release.name + ' </STRONG><A HREF="%s.html"' % release.name +
                                                                        '>[A]</A>',
                    Bold(Font(release.isOnDisk, color=release.isOnDiskC)),
                    Bold(Font(release.IRIXdebugfreeze, color=release.IRIXdebugfreezeC)),
                    Bold(Font(release.IRIXmaxoptfreeze, color=release.IRIXmaxoptfreezeC)),
                    Bold(Font(release.RH6debugfreeze, color=release.RH6debugfreezeC)),
                    Bold(Font(release.RH6maxoptfreeze, color=release.RH6maxoptfreezeC)),
                    Bold(Font(release.RH7debugfreeze, color=release.RH7debugfreezeC)),
                    Bold(Font(release.RH7maxoptfreeze, color=release.RH7maxoptfreezeC))]
            rowContents.append(line)

        relTable = Table('Available Frozen Releases',
            heading=['Release','On Disk','IRIX','IRIX<BR>maxopt','Red Hat 6','Red Hat ' +
                     '6<BR>maxopt','Red Hat 7','Red Hat 7<BR>maxopt'],

body_color=['#888888','#DDDDDD','#CCCCCC','#BBBBBB','#CCCCCC','#BBBBBB','#CCCCCC','#BBBBBB'],

heading_color=['#888888','#DDDDDD','#CCCCCC','#BBBBBB','#CCCCCC','#BBBBBB','#CCCCCC','#BBBBBB'],
            heading_align='center',
            column1_align='center',
            cell_align='center',
            body=rowContents)
        doc.append(relTable)

        doc.append(HR())
        doc.append_file(os.path.join(dataDir, 'info-txt.html'))
        doc.write(os.path.join(htmlDir, fileName))

    def errorsStatusPage(self, fileName, before=None, after=None, top=None, home=None):
        doc = SeriesDocument('RelStat.rc')
        doc.title = docTitle
        doc.subtitle = 'Errors Status'
        doc.banner = None
        doc.logo = (os.path.join(imageDir, 'd0logo.gif'),111,68)

        doc.prev = (os.path.join(imageDir, 'BTN_PrevPage.gif'),74,19)
        doc.next = (os.path.join(imageDir, 'BTN_NextPage.gif'),74,19)
        doc.top = (os.path.join(imageDir, 'BTN_ManualTop.gif'),74,19)
        doc.home = (os.path.join(imageDir, 'BTN_HomePage.gif'),74,19)
        doc.goprev = before
        doc.gonext = after
        doc.gotop = top
        doc.gohome = home

        rowContents = []
        for release in self.releases:
            if release.isOnDisk == 'yes':
                line = ['<STRONG>' + release.name + ' </STRONG><A HREF="%s.html"' % release.name
                                                                        + '>[A]</A>',
                        Bold(Font(release.isOnDisk, color=release.isOnDiskC)),
                        release.IRIXdebug.linkToLog,
                        release.IRIXmaxopt.linkToLog,
                        release.RH6debug.linkToLog,
                        release.RH6maxopt.linkToLog,
                        release.RH7debug.linkToLog,
                        release.RH7maxopt.linkToLog]
            else:
                line = ['<STRONG>' + release.name + ' </STRONG><A HREF="%s.html"' % release.name
                                                                        + '>[A]</A>',
                        Bold(Font(release.isOnDisk, color=release.isOnDiskC)),
                        Bold(Font(release.IRIXdebug.brokenPackages)),
                        Bold(Font(release.IRIXmaxopt.brokenPackages)),
                        Bold(Font(release.RH6debug.brokenPackages)),
                        Bold(Font(release.RH6maxopt.brokenPackages)),
                        Bold(Font(release.RH7debug.brokenPackages)),
```

```
                        Bold(Font(release.RH7maxopt.brokenPackages))]
            rowContents.append(line)

        relTable = Table('Number of broken packages',
            heading=['Release','On Disk','IRIX','IRIX<BR>maxopt','Red Hat 6','Red Hat ' +
                     '6<BR>maxopt','Red Hat 7','Red Hat 7<BR>maxopt'],

body_color=['#888888','#DDDDDD','#CCCCCC','#BBBBBB','#CCCCCC','#BBBBBB','#CCCCCC','#BBBBBB'],

heading_color=['#888888','#DDDDDD','#CCCCCC','#BBBBBB','#CCCCCC','#BBBBBB','#CCCCCC','#BBBBBB'],
            heading_align='center',
            column1_align='center',
            cell_align='center',
            body=rowContents)
        doc.append(relTable)

        doc.append(HR())
        doc.append_file(os.path.join(dataDir, 'info-txt.html'))
        doc.write(os.path.join(htmlDir, fileName))

    def annotationsPage(self, releaseName):
        annotationsFileName = releaseName + '.txt'
        annotationsPageName = releaseName + '.html'
        doc = SimpleDocument('RelStat.rc')
        doc.title = docTitle
        doc.append('<H3>%s</H3>' % releaseName)
        doc.append_file(os.path.join(annotationsDir, annotationsFileName))
        doc.write(os.path.join(htmlDir, annotationsPageName))

    def generate(self):
        t = time.clock()
        self.getReleaseStatus()
        self.overviewStatusPage('overview.html', None, 'build.html', 'overview.html',
                                                                      'status.html')
        self.buildStatusPage('build.html', 'overview.html', 'freeze.html', 'overview.html',
                                                                      'status.html')
        self.freezeStatusPage('freeze.html', 'build.html', 'errors.html', 'overview.html',
                                                                      'status.html')
        self.errorsStatusPage('errors.html', 'freeze.html', None, 'overview.html', 'status.html')
        for release in self.releases: self.annotationsPage(release.name)
        print 'Time to generate pages:', time.clock() - t, 'seconds'



if __name__ == '__main__':
    statusPages = StatusPages()
    statusPages.generate()
    print "Processing by ", __name__, " complete."
else:
    print "File ", __file__, "imported"
    pass
```

Appendix D

relanno.py

```
#! /usr/bin/env python

"""relanno.py appends a string or a file to a release annotations file."""

import os
import os.path
import sys
from HTMLgen.HTMLgen import *
from time import localtime, strftime


__author__ = 'Mariano Zimmler zimmler@fnal.gov'
__version__ = '0.1.0'

releaseStatusDir = sys.path[0]
annotationsDir = os.path.abspath(os.path.join(releaseStatusDir, 'annotations'))
htmlDir = os.path.abspath(os.path.join(releaseStatusDir, 'html'))

docTitle = 'Release Annotations Page (Version: %s)' % __version__

try:
    os.mkdir(annotationsDir)
    print "(relanno.py) - made annotations sub-directory"
except os.error:
    pass
    print "(relanno.py) - did not make annotations sub-directory (may already exist)."


def main():
    global releaseName
    flag = '-'
    annotation = '-'

    if len(sys.argv) < 4:
        print '\nERROR: Not enough arguments provided.'
        sys.exit(0)
    else:
        tReleasePattern1 = re.compile(r'^t\d\d\.\d\d\.\d\d')
        tReleasePattern2 = re.compile(r'^t\d\d\d-gcc')
        pReleasePattern = re.compile(r'^p\d\d\.\d\d\.\d\d')
        oReleasePattern = re.compile(r'^onl\d\d\.\d\d\.\d\d')
        if tReleasePattern1.match(sys.argv[1]) or tReleasePattern2.match(sys.argv[1]) or \
           pReleasePattern.match(sys.argv[1]) or oReleasePattern.match(sys.argv[1]):
            releaseName = sys.argv[1]
        else:
            print '\nERROR: Invalid release name: \'%s\'.' % sys.argv[1]
            sys.exit(0)
        if sys.argv[2] == '-f' or sys.argv[2] == '-m':
            flag = sys.argv[2]
            if flag == '-f':
                if os.path.isfile(sys.argv[3]):
                    annotation = sys.argv[3]
                else:
                    print '\nERROR: Invalid file name: \'s%\'.' % sys.argv[2]
                    sys.exit(0)
            else:
                annotation = sys.argv[3]
        else:
            print '\nERROR: Invalid argument \'s%\'.' % sys.argv[2]
            sys.exit(0)

    annotationsFileName = releaseName + '.txt'
    annotationsFile = os.path.join(annotationsDir, annotationsFileName)

    if flag == '-f':
```

```
        input = open(annotation, 'r')
        output = open(annotationsFile, 'a')
        output.write(strftime("<P>%a, %d %b %Y %H:%M:%S +0000<P>", localtime()))
        for line in input.xreadlines():
            output.write(line)
        input.close()
    elif flag == '-m':
        f = open(annotationsFile, 'a')
        f.write(strftime("<P>%a, %d %b %Y %H:%M:%S +0000<P>", localtime()))
        f.write(annotation)
        f.close()
    else:
        pass

def updateAnnotationsPage(releaseName):
    annotationsFileName = releaseName + '.txt'
    annotationsPageName = releaseName + '.html'
    doc = SimpleDocument('RelStat.rc')
    doc.title = docTitle
    doc.append('<H3>%s</H3>' % releaseName)
    doc.append_file(os.path.join(annotationsDir, annotationsFileName))
    doc.write(os.path.join(htmlDir, annotationsPageName))



if __name__ == '__main__':
    main()
    updateAnnotationsPage(releaseName)
    print "Processing by ", __name__, " complete."
else:
    print "File ", __file__, "imported"
    pass
```

Appendix E

update_relstat_err.py

```
source /usr/local/etc/setups.csh
setup n32
setup D0RunII $1
srt_setup SRT_QUAL=$2
cd $SRT_PUBLIC_CONTEXT
source ${SRT_PUBLIC_CONTEXT}/D0reltools/d0setwa.csh
find_broken.sh
```